10/ 028 010

Cort

10/028,010

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	**Steven R. Soltis**	Examiner:	**Chirag R. Patel**
Patent No.:	**7,165,096**	Group Art Unit:	**2141**
Filing Date:	**December 21, 2001**	Docket No.:	**2451**
Title	**Improved Storage Area Network File System**		

Date of Deposit: 5-22-07
I hereby certify that this paper is being deposited in the United States Postal Service, as first class mail, in an envelope addressed to: Commissioner for Patents, PO Box 1450, Alexandria, VA 22313-1450

Signature: Mary S. Keller
Printed Name: Mary S. Keller

REQUEST FOR CERTIFICATE OF CORRECTION

Commissioner for Patents
Alexandria, VA 22313

Errors appear in this patent. The errors are Patent Office errors and not the Applicant. Correction thereof does not involve such changes in the patent as would constitute new matter or would require reexamination. Attached hereto is a proposed Certificate of Correction set forth on the PTO/SB/44 form.

The errors occurred in the original patent specification on page 1, Preliminary Amendment (12/21/2001), line 11-12; sheet 2 of 3, List of References cited by applicant and considered by examiner (09/19/2005), entry 9, line 1 (Other Prior Art-Non Patent Literature Documents), page 1, Preliminary Amendment (12/21/2001), lines 11-12, page 18, Specification (12/21/2001) lines 26-27 and 29-30; page 3, Claims (07/10/2006), Claim 14, line 1, and page 8, Claims (07/10/2006), Claim 42, line 2.

Please send the certificate to:

Daniel A. Tysver
Beck & Tysver, P.L.L.C.
2900 Thomas Avenue S., Suite 100
Minneapolis, MN 55416

Since the errors are the responsibility of the Patent Office, we have not enclosed a fee. Any fee deficiency associated with this transmittal may be charged to Deposit Account 500-246.

Respectfully submitted,
DATAPLOW, INC.
By its attorneys:

Date: May 22, 2007

Daniel A. Tysver
Registration No. 35,726
Beck & Tysver, P.L.L.C.
2900 Thomas Ave., #100
Minneapolis, MN 55416
Telephone: (612) 915-9634
Fax: (612) 915-9637

PTO/SB/44 (04-04)
Approved for use through 04/30/2007. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 7,165,096

DATED : 06/16/2007

INVENTOR(S) : Steven R. Soltis

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

First Page, Column 1
Line 11
Above field (51) insert--
Related U.S. Application Data
Provisional application No. 60/257,446, filed on Dec. 22, 2000.--.

Page 2
Column 2 (Other Publications)
Line 6 Delete "Spite" and insert -- Sprite--, therefor.

Column 1
Line 3
Above FIELD OF THE INVENTION insert--
CROSS-REFERENCE TO RELATED APPLICATIONS
This application claims the benefit of Provisional application No. 60/257,446, filed on Dec. 22, 2000. - - .

Column 11
Lines 49-50 Delete "writeserialization" and insert--write-serialization--, therefor.
Line 54 Delete "readintensive" and insert --read-intensive--, therefor.

Column 20
Line 37 In Claim 14, delete "mode" and insert--inode --, therefor.

Column 23
Line 38 In Claim 42, delete "mode" and insert --inode--, therefor.

MAILING ADDRESS OF SENDER:

PATENT NO. 7,165,096

Beck & Tysver, P.L.L.C.
2900 Thomas Avenue S., Suite 100
Minneapolis, MN 55416

No. of additional copies
⇨

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

US007165096B2

(12) **United States Patent**
Soltis

(10) **Patent No.: US 7,165,096 B2**
(45) **Date of Patent: Jan. 16, 2007**

(54) **STORAGE AREA NETWORK FILE SYSTEM**

(75) Inventor: **Steven R. Soltis**, Rochester, MN (US)

(73) Assignee: **Data Plow, Inc.**, Rochester, MN (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 788 days.

(21) Appl. No.: **10/028,010**

(22) Filed: **Dec. 21, 2001**

(65) **Prior Publication Data**

US 2002/0083120 A1 Jun. 27, 2002

(51) **Int. Cl.**
G06F 15/16 (2006.01)
G06F 15/173 (2006.01)

(52) **U.S. Cl.** 709/217; 709/238; 707/1

(58) **Field of Classification Search** 709/216, 709/217, 238; 707/1
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,043,876	A	8/1991	Terry	
5,202,971	A *	4/1993	Henson et al.	707/8
5,561,799	A *	10/1996	Khalidi et al.	707/200
5,652,913	A	7/1997	Crick et al.	
5,668,958	A	9/1997	Bendert et al.	
5,740,230	A *	4/1998	Vaudreuil	379/88.22
5,758,342	A *	5/1998	Gregerson	707/10
5,764,972	A	6/1998	Crouse et al.	
5,802,366	A *	9/1998	Row et al.	709/250
5,828,876	A	10/1998	Fish et al.	
5,909,540	A	6/1999	Carter et al.	
5,931,918	A	8/1999	Row et al.	
5,933,603	A	8/1999	Vahalia et al.	
5,978,773	A	11/1999	Hudetz et al.	

(Continued)

OTHER PUBLICATIONS

T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang, "Serverless Network File System," Proceedings of the Fifteenth ACM Symposium on Operating System Principles, 1995, The Association for Computer Machinery (ACM) Press, New York, NY USA.

(Continued)

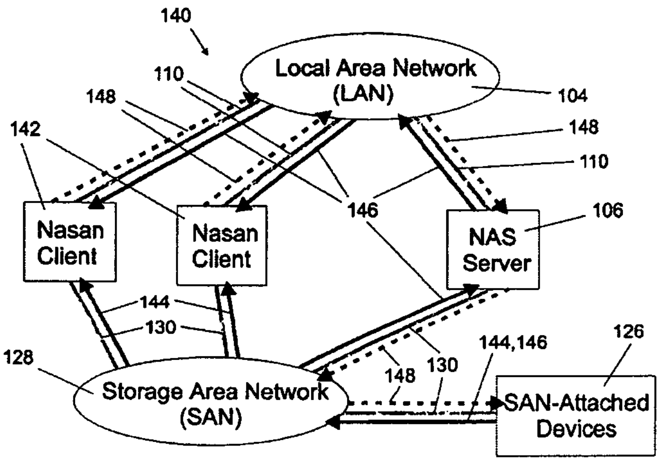
Primary Examiner—Rupal Dharia
Assistant Examiner—Chirag R Patel
(74) *Attorney, Agent, or Firm*—Beck & Tysver, P.L.L.C.

(57) **ABSTRACT**

A shared storage distributed file system is presented that provides applications with transparent access to a storage area network (SAN) attached storage device. This is accomplished by providing clients read access to the devices over the SAN and by requiring most write activity to be serialized through a network attached storage (NAS) server. Both the clients and the NAS server are connected to the SAN-attached device over the SAN. Direct read access to the SAN attached device is provided through a local file system on the client. Write access is provided through a remote file system on the client that utilizes the NAS server. A supplemental read path is provided through the NAS server for those circumstances where the local file system is unable to provide valid data reads.

Consistency is maintained by comparing modification times in the local and remote file systems. Since writes occur over the remote file systems, the consistency mechanism is capable of flushing data caches in the remote file system, and invalidating metadata and real-data caches in the local file system. It is possible to utilize unmodified local and remote file systems in the present invention, by layering over the local and remote file systems a new file system. This new file system need only be installed at each client, allowing the NAS server file systems to operate unmodified. Alternatively, the new file system can be combined with the local file system.

61 Claims, 12 Drawing Sheets



U.S. PATENT DOCUMENTS

5,987,621	A	11/1999	Duso et al.	
6,442,682	B1 *	8/2002	Pothapragada et al.	713/1
2001/0037406	A1 *	11/2001	Philbrick et al.	709/250
2002/0112022	A1 *	8/2002	Kazar et al.	709/217

OTHER PUBLICATIONS

M. Devarakonda, A. Mohindra, J. Simoneaux, and W. Tetzlaff, "Evaluation of Design Alternatives for a Cluster File System," 1995 USENIX Technical Conference on UNIX and Advanced Computing Systems, Jan. 1995, pp. 35-46, USENIX Association, Berkeley, CA USA.

G. Gibson, D. Nagle, K. Amiri, F. Chang, H. Gobioff, E. Riedel, D. Rochberg, and J. Zelenka, "File Server Scaling with Network-Attached Secure Disks," Proceedings of the ACM International Conference on Measurements and Modeling of Computer Systems, Jun. 1997, The Association for Computer Machinery (ACM) Press New York NY USA.

G. Gibson, D. Nagle, K. Amiri, F. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka, "Filesystems for Network-Attached Secure Disks," Carnegie Mellon University Technical Report CMU-CS-97-118, Jul. 1997, Carnegie Mellon University, Pittsburgh, PA USA.

J. Heidemann and G. Popek, "File System Development with Stackable Layers," ACM Transaction on Computer Systems, 1994, pp. 58-89, The Association for Computer Machinery (ACM) Press, New York, NY USA.

Y. Khalidi and M. Nelson, "Extensible File Systems in Spring", Sun Microsystems Laboratories Technical Report TR-93-18, Sep. 1993, Sun Microsystems Laboratories, Inc., Mountain View, CA USA.

S. Kleiman, "Vnodes: An Architecture for Multiple File System Types in Sun UNIX," Proceedings of the Summer USENIX Conference, Jun. 1986, pp. 238-247, USENIX Association, Berkeley, CA USA.

K. Matthews, "Implementing a Shared File System on a HIPPI Disk Array," Fourteenth IEEE Symposium on Mass Storage Systems, 1995, pp. 77-88, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ USA.

J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch, The Spite Network Operating System, IEEE Computer, Feb. 1988, pp. 23-36, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ USA.

B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS Version 3: Design and Implementation," Proceedings of the Summer USENIX Conference, 1994, USENIX Association, Berkeley, CA.

D. Rosenthal, "Evolving the Vnode Interface," Proceedings of the Summer USENIX Conference, Jun. 1990, pp. 107-117, USENIX Association, Berkeley, CA USA.

R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System," Proceedings of the Summer USENIX Conference, 1985, pp. 119-130, USENIX Association, Berkeley, CA USA.

M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access," IEEE Computer, May 1990, pp. 9-20, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ USA.

M. Satyanarayanan, "Code: A Highly Available File System for a Distributed Workstation Environment," Proceedings of the Second IEEE Workshop on Workstation Operating Systems, Sep. 1989, Institute of Electrical and Electronics Engineers, Inc., Piscataway, NJ USA.

S. Soltis, The Design and Implementation of a Distributed File System base on Shared Network Storage. PhD Thesis, University of Minnesota, 1997, University of Minnesota, Minneapolis, MN USA.

D. Wiltzius and K. Minuzzo, "Network-attached peripherals (NAP) for HPSS/SIOF," Lawrence Livermore National Laboratory Technical Report, 1995. Available at http://www.llnl.gov/liv_comp/siof/siof_nap.html, Livermore, CA USA.

* cited by examiner

STORAGE AREA NETWORK FILE SYSTEM

FIELD OF THE INVENTION

The present invention relates general to computer file systems. More specifically, the present invention involves a distributed file system that transfers data using both network attached storage (NAS) and storage area network (SAN) protocols.

BACKGROUND OF THE INVENTION

File Systems

The term "file system" refers to the system designed to provide computer application programs with access to data stored on storage devices in a logical, coherent way. File systems hide the details of how data is stored on storage devices from application programs. For instance, storage devices are generally block addressable, in that data is addressed with the smallest granularity of one block; multiple, contiguous blocks form an extent. The size of the particular block, typically 512 bytes in length, depends upon the actual devices involved. Application programs generally request data from file systems byte by byte. Consequently, file systems are responsible for seamlessly mapping between application program address-space and storage device address-space.

File systems store volumes of data on storage devices. The term "volume" refers to the collection of data blocks for one complete file system instance. These storage devices may be partitions of single physical devices or logical collections of several physical devices. Computers may have access to multiple file system volumes stored on one or more storage devices.

File systems maintain several different types of files, including regular files and directory files. Application programs store and retrieve data from regular files as contiguous, randomly accessible segments of bytes. With a byte-addressable address-space, applications may read and write data at any byte offset within a file. Applications can grow files by writing data to the end of a file; the size of the file increases by the amount of data written. Conversely, applications can truncate files by reducing the file size to any particular length. Applications are solely responsible for organizing data stored within regular files, since file systems are not aware of the content of each regular file.

Files are presented to application programs through directory files that form a tree-like hierarchy of files and subdirectories containing more files. Filenames are unique to directories but not to file system volumes. Application programs identify files by pathnames comprised of the filename and the names of all encompassing directories. The complete directory structure is called the file system namespace. For each file, file systems maintain attributes such as ownership information, access privileges, access times, and modification times.

File systems often utilize the services of operating system memory caches known as buffer caches and page caches. These caches generally consist of system memory buffers stored in volatile, solid-state memory of the computer. Caching is a technique to speed up data requests from application programs by saving frequently accessed data in memory for quick recall by the file system without having to physically retrieve the data from the storage devices. Caching is also useful during file writes; the file system may write data to the memory cache and return control to the application before the data is actually written to non-volatile storage. Eventually, the cached data is written to the storage devices.

The state of the cache depends upon the consistency between the cache and the storage devices. A cache is "clean" when its contents are exactly the same as the data stored on the underlying storage devices. A cache is "dirty" when its data is newer than the data stored on storage devices; a cache becomes dirty when the file system has written to the cache, but the data has not yet been written to the storage devices. A cache is "stale" when its contents are older than data stored on the storage devices; a cache becomes stale when it has not been updated to reflect changes to the data stored on the storage devices.

In order to maintain consistency between the caches and the storage devices, file systems perform "flush" and "invalidate" operations on cached data. A flush operation writes dirty cached data to the storage devices before returning control to the caller. An invalidation operation removes stale data from the cache without invoking calls to the storage devices. File systems may flush or invalidate caches for specific byte-ranges of the cached files.

Many file systems utilize data structures called inodes to store information specific to each file. Copies of these data structures are maintained in memory and within the storage devices. Inodes contain attribute information such as file type, ownership information, access permissions, access times, modification times, and file size. Inodes also contain lists of pointers that address data blocks. These pointers may address single data blocks or address an extent of several consecutive blocks. The addressed data blocks contain either actual data stored by the application programs or lists of pointers to other data blocks. With the information specified by these pointers, the contents of a file can be read or written by application programs. When an application programs write to files, data blocks may be allocated by the file system. Such allocation modifies the inodes.

Additionally, file systems maintain information, called "allocation tables", that indicate which data blocks are assigned to files and which are available for allocation to files. File systems modify these allocation tables during file allocation and de-allocation. Most modem file systems store allocation tables within the file system volume as bitmap fields. File systems set bits to signify blocks that are presently allocated to files and clear bits to signify blocks available for future allocation

The terms real-data and metadata classify application program data and file system structure data, respectively. In other words, real-data is data that application programs store in regular files. Conversely, file systems create metadata to store volume layout information, such as inodes, pointer blocks, and allocation tables. Metadata is not directly visible to applications. Metadata requires a fraction of the amount of storage space that real-data occupies and has significant locality of reference. As a result, metadata caching drastically influences file system performance.

Metadata consistency is vital to file system integrity. Corruption of metadata may result in the complete destruction of the file system volume. Corruption of real-data may have bad consequences to users but will not affect the integrity of the whole volume.

I/O Interfaces

I/O interfaces transport data among computers and storage devices. Traditionally, interfaces fall into two categories: channels and networks. Computers generally communicate with storage devices via channel interfaces. Channels predictably transfer data with low-latency and high-bandwidth

The I/O interface links **130** connect to the SAN **128**, which consists of network components such as routers, switches, and hubs. The SAN **128** may also include components that perform storage virtualization, caching, and advanced storage management functions. The SAN devices **126** are block and object-addressable, non-volatile storage devices. The SAN devices **126** may be part of a SAN appliance **136** or dedicated storage devices attached to the SAN **128**.

The primary read data-path **144** of Nasan is similar to the read data-path **132** of prior art SAN environments **120**, whereas the secondary read data-path **146** is similar to the read data-path **114** of prior art NAS environments **100**. The majority of read transfers take place over the primary data-path **144**, which passes from the SAN devices **126**, through the SAN **128**, directly to the Nasan clients **142**. The primary data-path **144** takes full advantage of high-speed SAN protocols. However, some read transfers follow the secondary data-path **146** and pass from the SAN-attached devices **126**, through the NAS server **106**, across the LAN **104**, en-route to the Nasan clients **142**. The state of the Nasan environment **140** dictates whether the primary data-path **144** or the secondary data-path **146** is used for read transfers.

The write data-path **148** of Nasan is similar to the write data-path of prior art NAS **116** with the difference being the Nasan write data-path **148** also includes the SAN **128**. The write data-path **148** begins at the Nasan clients **142** and passes through the LAN **104** to the NAS server **106**. The server **106**, in turn, writes across the SAN **128** to the SAN-attached devices **126**.

Due to high-speed SAN reads **144**, the Nasan file system significantly exceeds the file sharing performance and scalability of prior art NAS solutions. Although Nasan write performance is similar to prior art NAS write performance, Nasan reads are often ten times faster. Because read operations generally outnumber writes five to one, the performance improvement made to reads dramatically increases overall system throughput. Furthermore, by offloading reads from the NAS servers **106**, the Nasan file system substantially reduces server **106** workloads. With reduced workloads, servers **106** exhibit shorter response times, sustain more simultaneous file transfers, and support considerably larger throughputs than servers **106** supporting traditional NAS **100**.

The Nasan file system transfers read requests across the high-speed SAN **128** while serializing writes through a central NAS server **106**. This serialization leads to write transfer rates that are slower than reads; however, writeserialization facilitates extremely low-latency consistency management. Low-latency consistency enables Nasan clients **142** to efficiently transfer files of all sizes. Therefore, the Nasan file system is a general-purpose solution for readintensive workloads.

Nasan Layering

One embodiment of the Nasan file system utilizes a two-tiered layering scheme. Nasan software occupies the upper level, while non-modified local and remote file systems comprise the lower. The Nasan layer provides a management framework that facilitates data consistency and routes file requests to the appropriate lower level file system. All remaining file management functionality is derived from these lower layer file systems.

Referring to FIG. **5**, application programs **150** running on the Nasan client **142** make file requests to the Nasan file system software layer **152**. Nasan software **152** redirects read requests to the either the local file system level **154** or the client-side remote file system layer **156** and redirects write requests to the remote file system layer **156**. These lower layer file systems conduct the actual data management, transport, and storage.

The local file system **154** of the client provides the primary read data-path **144** for Nasan transfers. Because the clients **142** do not directly modify the volume stored on the SAN devices **126**, Nasan software **152** maintains low-latency consistency by simply invalidating stale caches of the local file system layer **154**.

The remote file system facilitates the secondary read data-path **146** as well as write access to files managed by the NAS server **106**. The Nasan client **142** passes file requests to the client-side remote file system layer **156**. In turn, the remote file system **156** on the client **142** transmits these requests via NAS protocols to the server-side remote file system layer **158** on the server **106**. The NAS server **106** completes the requests by reading data from or writing data through the local file system **155** of the server **106** to volumes stored on SAN-attached devices **126**. Write-serialization, through the NAS server **106**, enables low-latency consistency.

Components of the Preferred Embodiment

The components and protocols that form the environment **140** of the present invention range in price, performance, and compatibility. In the preferred embodiment, the interface links **110,130** that connect to the LAN **104** and to the SAN **128** may include Ethernet, InfiniBand, and Fibre Channel. Over these links **110,130** run a number of different network and channel protocols, including Internet Protocol (IP), SCSI-3, Virtual Interface (VI), iSCSI, FCIP, and iFCP. The NAS protocols used by the remote file system **156,158** include Network File System (NFS), Server Message Block (SMB), and Common Internet File System (CIFS). The present invention is not limited to these specific components and protocols.

Local File System Consistency

In general, local file systems perform extensive metadata and real-data caching. The only consistency management typically required of local file systems is periodic updates to on-disk data structures. Cached data is never invalidated because on-disk data is always assumed to be older than cached data.

Within a Nasan environment **140**, the NAS server **106** has read-write access to the local file system volume stored on SAN-attached disks **126**, while Nasan clients **142** have read-only access to this volume. Because the client local file systems **154** and the server local file systems **155** may not be designed to support SAN environments with multiple computers, Nasan software **152** must explicitly maintain data consistency between storage devices **126** and caches of the client local file system **154**.

Consistency Between Local and Remote File System Layers

Local **154** and remote **156** file systems utilize separate caches within client **142** main memories. After file writes, the remote file system **156** cache contains newer data than the local file system **154** cache. Nasan software **152** makes the local file system **154** cache consistent with the Nasan environment **140** by explicitly invalidating stale data within the cache.

Nasan software **152** has the option to read from the local file system **154** or the remote file system **156**. When reading from the primary data-path **144**, Nasan software **152** first determines if data is being cached by the client-side remote file system **156**. If data is cached, Nasan software **152** flushes the remote file system **156** cache and invalidates the locked, the Nasan client **142** reads the allocation tables from the SAN devices **126**, modifies the allocation tables, writes the tables to the SAN devices **126**, and then releases the file lock.

At step **418**, the file is fully allocated for the request range. The Nasan write function writes the real-data to the SAN-attached devices **126** via the SAN write data-path **326**. Once this real-data write completes, at step **420**, the modified on-disk inode is written by the client **142** to the SAN-attached devices **126** and the file lock is released by issuing an unlock request to the client-side remote file system **156**. The remote file system **156** passes the unlock request to the server **106** which forwards the unlock request to the server-side Nasan file system **324**. After the file lock is released, the Nasan write operation completes.

The invention is not to be taken as limited to all of the details thereof as modifications and variations thereof may be made without departing from the spirit or scope of the invention. For instance, the present invention was described and shown with the SAN and LAN networks appearing as separate, physical networks. However, as is well known in the prior art, it is possible to send SAN protocols and LAN protocols over the same physical network. The two networks are distinguishable by the protocols that are used to communicate between nodes on the network. In addition, although it is not shown in the drawings, it would be possible to use a client computer in the present invention as a file server that serves file requests from other computers. These other computers would likely have no access to the storage area network, but would have the ability to send file requests to the client computer of the present invention over a local area network. Because many such modifications and variations are present, the scope of the present invention is not to be limited to the above description, but rather is to be limited only by the following claims.

What is claimed is:

1. A file system on a computer handling file read and file write requests to a SAN-attached storage device comprising:

a) a local component that communicates with the SAN-attached storage device over a storage area network and that interprets metadata stored on the SAN-attached storage device;

b) a NAS server that communicates with the SAN-attached storage device over a storage area network;

c) a remote component that communicates with the NAS server over a local area network;

d) an upper level component that communicates with application programs, the upper level component submitting all file write requests to the remote component and submitting at least some file read requests to the local component.

2. The file system of claim 1), wherein each of the components are separate file systems in which the upper level component file system is layered above the local component file system and the remote component file system.

3. The file system of claim 2) utilizing an installable file system interface to facility layering between the file systems.

4. The file system of claim 3) is the Virtual File System interface.

5. The file system of claim 2), wherein the remote component file system utilizes a protocol chosen from among the following set: Network File System, Server Message Block, and Common Internet File System.

6. The file system of claim 1), wherein the upper level component and the local component are merged into a single file system, and further wherein the remote component is a separate file system layered under the single file system containing the upper level and local components.

7. The file system of claim 1), wherein the upper level component submits all read requests to the local component.

8. The file system of claim 1), wherein the upper level component submits read requests above a certain size to the local component, and read requests below the certain size to the remote component.

9. The file system of claim 1), wherein the upper level component submits read requests to the local component if the file size is above a certain size, and the upper level component submits read requests to the remote component if the file size is below the certain size.

10. The file system of claim 1), wherein the upper level component submits all read request to the local component except where the local component is not capable of properly retrieving data requested in the read request, in which case the upper level component submits the read request to the remote component.

11. The file system of claim 10), wherein the upper level component determines whether the local component is capable of properly retrieving data requested in the read request by comparing modification times for a file indicated in the read request as retrieved from the remote component and the local component.

12. The file system of claim 11), further wherein the upper level component determines whether the local component is capable of properly retrieving data requested in the read request by comparing modification times for a directory indicated in the read request as retrieved from the remote component and the local component.

13. The file system of claim 12), wherein the directory modification times are compared during a lookup function, with the results stored in an inode structure for the higher-level component.

14. The file system of claim 13), wherein the mode structure for the higher-level component includes the following:

a) a handle pointing to a file vnode for the remote component;

b) a handle pointing to a file vnode for the local component;

c) a remote modification time indicating the modification time returned by the remote component; and

d) a local modification time indicating the modification time returned by the local component.

15. The file system of claim 1), wherein the file system is capable of handling additional file requests, with file requests that alter data on the SAN-attached storage device being treated similar to the write requests, and file requests that do not alter data on the SAN-attached storage device being treated similar to the read requests.

16. The file system of claim 1), further comprising a file server component capable of receiving and responding to file requests from other computers that are connected to the local area network but not connected to the storage area network.

17. A network of connected computing devices comprising:

a) a local area network;

b) a storage area network;

c) a SAN-attached device attached to the storage area network;

d) a server computer attached to the local area network and the storage area network; the server computer receiving file requests across the local area network and

iii) determining whether file requests that do not modify the SAN-attached device can be fulfilled through the local file system;

iv) treating file requests that do not modify the SAN-attached device and can be fulfilled through the local file system as local requests; and

v) treating the remaining file requests as remote requests.

39. The method of claim 37), wherein the step of determining whether file requests can be fulfilled through the local file system further comprises:

(1) determining a file involved in the file requests;

(2) retrieving a modification time for the file from the local file system;

(3) retrieving a modification time for the file from the remote file system; and

(4) comparing the local and remote modification times.

40. The method of claim 39), wherein the step of determining whether file requests can be fulfilled through the local file system further comprises:

(5) invalidating the local file system cache when the two modification times are not identical.

41. The method of claim 39), wherein the step of determining whether file requests can be fulfilled through the local file system further comprises:

(6) obtaining a new modification time from the local file system after the local file system cache has been invalidated;

(7) comparing the new local modification time with the remote modification time in a second comparison;

(8) treating the file request as a local request if the second comparison finds the modification times to be identical; and treating the file request as a remote request if the second comparison does not find the modification times to be identical.

42. The method of claim 41), further comprising the step of performing a lookup function to create an inode for the file, the mode having a remote handle pointing to a remote vnode for the file and a local handle pointing to a local vnode for the file, the remote and local vnodes being used to identify the file in the remote and local file systems, respectively.

43. The method of claim 42), wherein the step of performing a lookup function compares modification times for the directory containing the file in both the remote and local file systems to determine whether the local file system access should be allowed.

44. The method of claim 43), wherein a lookup routine is performed for the file for both the local and remote file systems, and further wherein, if the remote file system does not find the file, a file not found result is returned to the application.

45. The method of claim 44), wherein if the local file system does not find the file, the remote file system is used to service file requests.

46. A network of connected computing devices comprising:

a) a local area network;

b) a storage area network;

c) a SAN-attached device attached to the storage area network;

d) a server computer attached to the local area network and the storage area network; the server computer receiving file requests across the local area network and further storing and retrieving data on the SAN-attached device via the storage area network; and

e) at least one client computer attached to the local area network and the storage area network; the client computer having:

i) a remote component in communication with and making file requests to the server computer over the local area network;

ii) a local component in communication with and making metadata and real-data requests to the SAN-attached device over the storage area network;

iii) an upper level component serving file requests from an application program operating on the client computer, the upper level component dividing the file requests from the application program between the remote component and the local component.

47. The network of claim 46), wherein the upper level component submits all file requests having data sizes above a certain size to the SAN-attached device over the storage area network via the local component and all file requests having data sizes below a certain size to the server computer over the local area network via the remote component.

48. The network of claim 46), wherein the upper level component submits all file requests to the SAN-attached device over the storage area network via the local component if the file size is above a certain size, and the upper level component submits all file requests to the server computer over the local area network via the remote component if the file size is below the certain size.

49. The network of claim 46), wherein file requests comprise write requests and read requests, and the upper level component submits all write requests to the server computer over the local area network via the remote component, and at least some of the read requests to the SAN-attached device over the storage area network via the local component.

50. The network of claim 49), wherein the upper level component submits all read requests having data sizes above a certain size to the SAN-attached device via the local component, and further wherein the upper level component submits all read requests having data sizes below a certain size to the server computer via the remote component.

51. The network of claim 49), wherein the upper level component submits all read requests to the SAN-attached device via the local component if the file size is above a certain size, and further wherein the upper level component submits all read requests to the server computer via the remote component if the file size is below the certain size.

52. The network of claim 49), wherein the upper level component submits all read requests through the local component except where the local file system is not capable of properly retrieving data requested in the read request, in which case the upper level component submits the read request to the remote component.

53. The network of claim 52), wherein the upper level component determines whether the local component is capable of properly retrieving data requested in the read requests by comparing modification times received from the local component with modification times received from the remote component.

54. The network of claim 49), further comprising a plurality of additional client computers, wherein, in relation to the client computers, only the server computer is granted write access to the SAN-attached device and further wherein all write requests from the client computers are routed via the remote component in the client computers to the server computer.

55. The file system of claim 46), wherein the client computer further has a file server component that receives